Project 1:

Title:
Uno Card Game

Class:
CSC 17A - 43396

Due Date:
May 16th, 2021

Author:
Aamir Khan

Introduction and Gameplay:

Title: Uno Card Game

Goal:

Each player begins with a certain number of cards that make up their deck. The goal of the game is to use up all of your cards before your opponent does so. To use these cards, players are expected to match their cards with the card currently on the card pile. Players can even sabotage each other by drawing special cards that we will talk about in the next section. Overall, this is a fun and simple game that can be played by kids and adults of all ages.

The How-To:

Cards can be matched in a variety of ways, and that is what we'll be going over shortly. A random card will be placed in the card pile between two players. Players take turns matching whatever card is currently on the card pile. Cards can be matched either by colors, values, or using special cards such as the Wild card, the Draw 1 card, or the Draw 2 card. To match by color, the cards need to have matching colors only. To match by value, the cards just need to have the same face value [Between 1-9]. For example, if we have a card that is green and has a value of 5 while the other card is red and also has a value of 5, these cards can be matched by value. As for drawing special cards, a Wild card may be used at any time throughout the game. This card can be used to set a new color to match, chosen by the one who used the card. A Draw 1 card forces the opponent to draw 1 card and add it to their deck of cards. A Draw 2 card forces the opponent to draw 2 cards. Draw 1 and 2 cards may only be used when the colors match or when the card that needs to be matched is also a Draw 1 or 2 card. For example, if we have a green Draw 1 and the other card is a blue Draw 1, these cards can be matched because they're both Draw 1's.

Summary:

The purpose of this project was to build upon my experience and implement fairly new concepts that we learned up until now in CSC 17A. These concepts may include: pointers, structures, nested structures, binary files, and more. The game utilizes constructs ranging from Dependent If's to Switch statements (And a Do-While), and I worked especially hard to get the code in main to be concise and clear-cut. I made sure of this by packaging most, if not all, of my operations in functions to ensure code reusability and convenience. The code itself isn't too lengthy, but it gets the job done, and a fully capable game of Uno can be played with two players in this program.

Stages of Development:

Version 1: CardGenerator

For the very first version of my game, I worked out a simple card generator. I had numbers drawn that would determine the color of our card as well as numbers that would determine our card's value. From there, I generated several random cards to see if the basics were working and whether I could proceed to add a few more ideas to the game.

Version 2: CardGenerator

For this version, I focused on generating Draw 1's, Draw 2's, and Wild cards. I added the three special cards into the mix of what I had already included in version 1. Besides this addition, the changes were slight, and not much was added to this version.

Version 3: CardStructure

This is where the program really began because this was the version where I had a structure created to represent a single Uno card. Everything that I had developed up until now would be stored in this structure. The structure consisted of three members: the color, the value, and the description if the card was one of the special cards. I had everything in the previous versions brought over with the addition of the information now being stored in an UnoCard structure.

Version 4: CardStructure

In this version, I put together an array of structure variables from the structure I previously created in version 3. Moreover, I added a set card function which would generate a random card and return it, so that it could be stored in a structure variable. This set card function would be then used for future versions any time I needed a card to be set or randomly generated. If I hadn't already mentioned, the set card function utilizes switch statements to create these random cards. For now, the special cards all have no color, but later on, we'll be adding colors to the Draw 1's and Draw 2's because they are supposed to have colors. The only card that doesn't have a real color is the Wild card because it is used to set a new color to match.

Version 5: CardMatching

From this version onward would be where the real game comes into play. It was in this version that I got matching by color and value to start working. I also threw in a display function which would display the contents of a UnoCard structure variable. On the side, I created a clear card function to show that the contents of a card were cleared once it was used. In other words, this function was meant to show that the card was being discarded once it was used to match a card. This function would later be removed and combined with a copy card function.

Version 6: CardMatching

As I previously mentioned, version 5 was when I began working out the logic for Uno. In this version, I made sure that the player was able to match multiple times and have more than just one go at matching cards. This was to ensure that the logic was functional and that we could take as many turns as we wanted. I also had a copy card function created that copied the contents of the last card in our structure array into the empty spot of the card that was just used. For example, if we had 7 cards and we used card 3, card 7's contents would be copied over to card 3's spot. I did this to make sure that numbering would still work and also to avoid having an empty slot in our array. The copy card function copies the contents of the first card into the second, and it also clears the contents of the second card which is why I had the clear card function combined with this function.

Version 7: CardMatching

In this version, the logic became much more complicated. I worked out just how the Wild card would work and what would happen if it were used to match a card. I also created a draw card function that would be used to draw cards in the case a Draw 1 or Draw 2 card was used. The draw card function takes the structure array, the current number of cards, and the number of cards that need to be drawn. From there, a simple for loop is called that uses the set card function to generate random cards. For now, we had the player draw cards if this were the case, but in future versions, the opponent would be the one drawing cards. Throughout the program, the copy card function is used to prevent jumps in the number of cards we have and to fill in empty spots whenever a card is used.

Version 8: CardMatching

This was a version meant more for practice than it was for actually getting the game together. Instead of using a static array, I used a dynamic array of structure variables for the player's deck of cards. I also threw in some pointer notation to work alongside the array notation I already had. The nice thing about this version is that the only real thing I had to change was passing the array in as a pointer instead of the array itself. In the next version, the real game would begin and the fun would start!

Version 9: GameSet

In this version, I had an opponent added into the mix, making it more of a real Uno game than just working out the mechanics. I also threw most of my code into a function for taking turns. That way, I would save space and create a cleaner program in main. In doing so, I was able to cut my program in half considering how long it would've been had I not created this function for taking turns. The function accepts the current player's deck, opponent's deck, the card to match, the player's number of cards, and the opponent's number of cards. If a Draw 1 or Draw 2 card were ever used, the opponent would now be the one drawing the cards. We pass the number of cards by reference because the function modifies both values when necessary and these changes need to be kept. An idea for the next version is to create a structure that would represent a player, so that we could implement a structure within a structure. This would also cut down on some arguments that we pass into the take turn function because the player structure would have both the number of cards a player has as well as their deck of cards.

Version 10: GameSet

In this version, I put together an UnoPlayer structure that stored the number of cards a player has and a pointer to an array of UnoCard structure variables that would represent their deck of cards. Besides changing the syntax from the previous version, I managed to translate everything over quite nicely. In creating this player structure, I was able to reduce the number of arguments that needed to be passed to the take turn function. It would now accept the current player's structure, the opponent's structure, and the card to match. I then used a bool to determine when a player had used all of their cards. Once this condition was met, the program would exit the do-while loop that had been used to work out the game, and the winner would be announced. I also threw

in a file for output on the side that listed the rules of Uno and a little about what it's all about. Hopefully by the next version, I could add binary files into the mix, but it isn't quite clear how I would want to use binary files in my program. I could store the number of cards each player ends up with or I could store some information on the cards we began with and read in that information by the end of the program. Besides the requirements that I need to add in, this version has a fully functional game of Uno that can be played with two players.

Version 11: GameSet

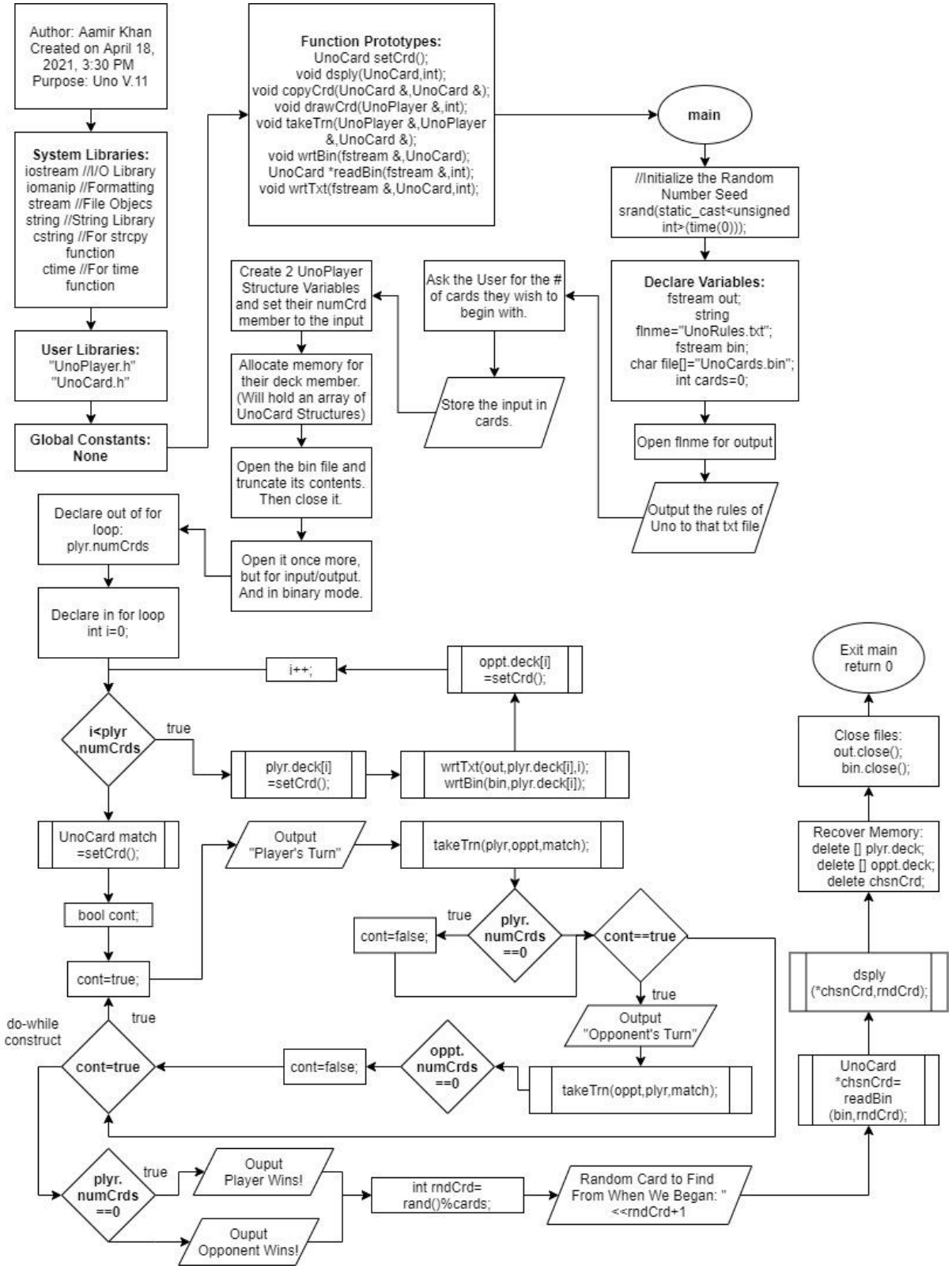
In this version, I added a binary file into my program. I used this file for both input and output. The plan was to write all of the player's cards to the file and read in the data of a randomly chosen card at the very end of the program. That way, we could read in data on one of the cards the player had at the very beginning of the game before any cards were even matched. This would demonstrate data being written to the file at the beginning, and binary data being read at the end of the program. In addition, I had the data read and written using functions that took the fstream objects by reference and the card that was being written to the file or read from the file. I also made sure to include a write text function which wrote each of the player's cards to the text file we had from before. This version was the final product of my game.

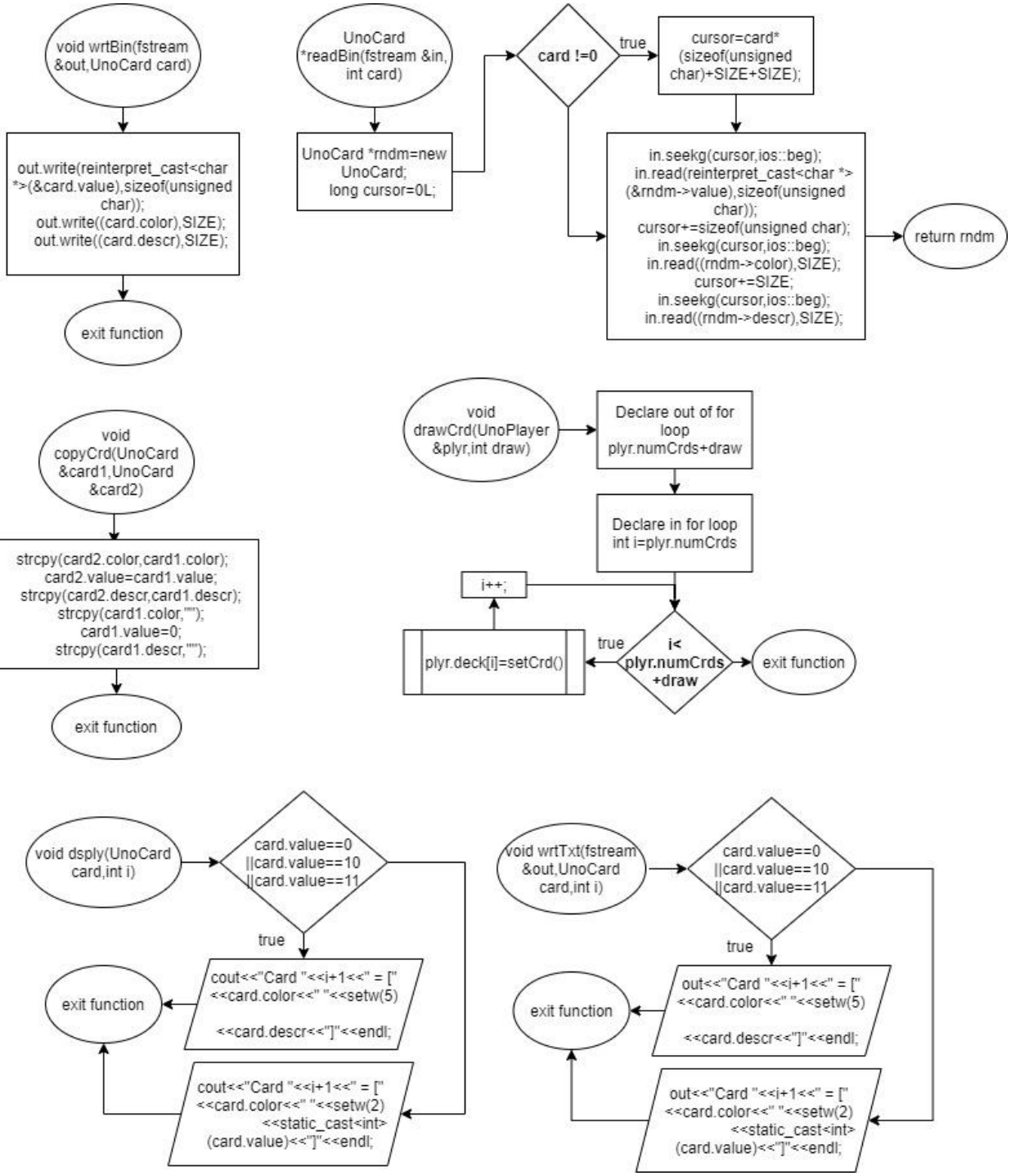
Project 1: Pseudo-Code

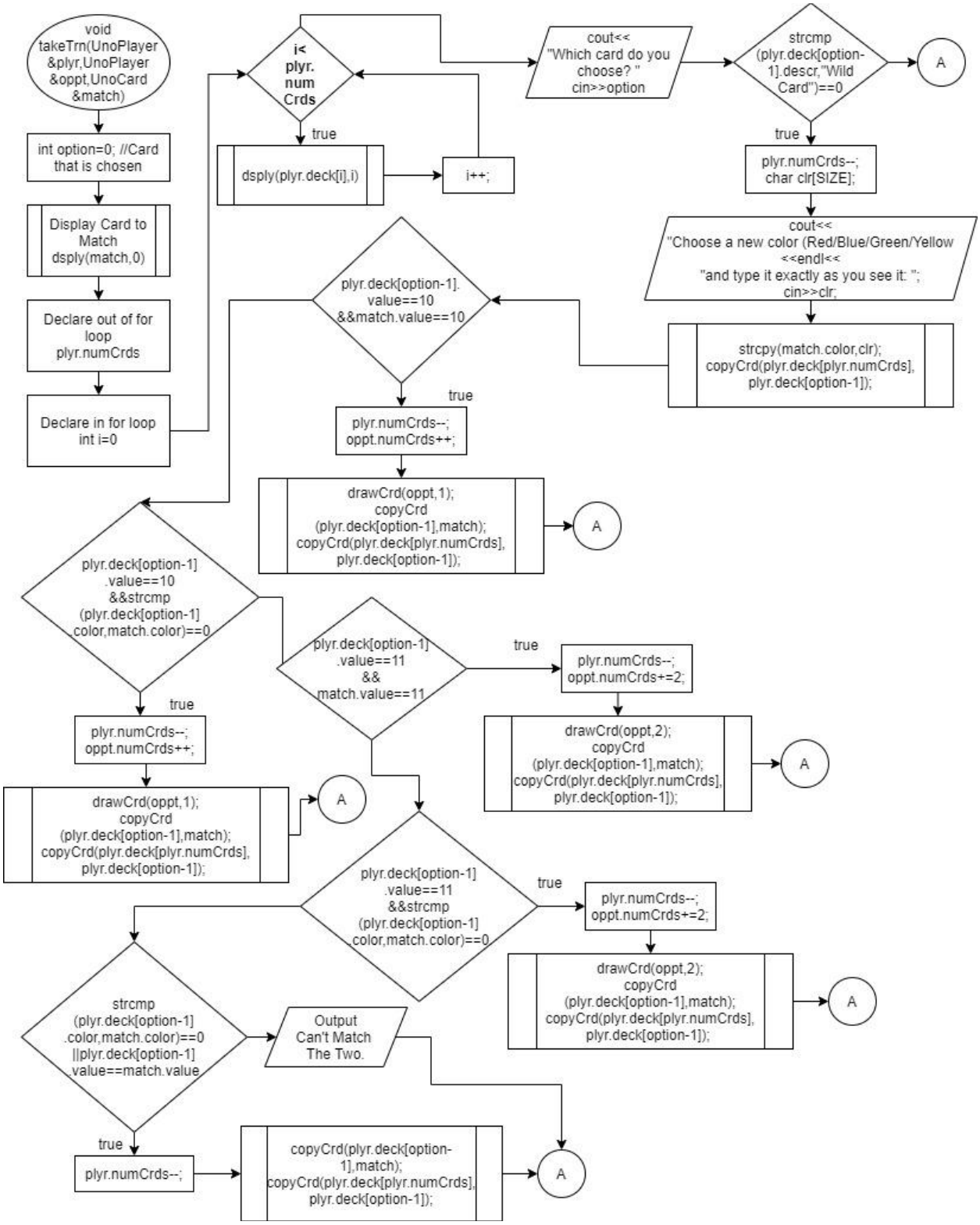
```
/*
* File:   main.cpp
 * Author: Aamir Khan
 * Created on April 18, 2021, 3:30 PM
 * Purpose:  Uno Version 11
 */


//System Libraries
//I/O Library
//Formatting Library
//String Library
//File Library
//Rand # Generator
//Time set to seed
//Namespace std of system libraries


//User Libraries
//Include UnoPlayer File (Contains UnoCard File)

//Global Constants
//No Global Constants Used

//Function Prototypes
//List all my prototypes

//Main -> Execution Begins Here

        //Initialize the Random # Generator Seed

        //Declare Variables (fstream objects + file names)

        //Send the rules to UnoRules.txt

        //Start the game by asking the player for the # of cards they wish to start with

        //Create 2 UnoPlayer variables and set their numCrds to the input above

        //Allocate memory for both UnoPlayer Objects

        //Clear the contents of the bin file
```

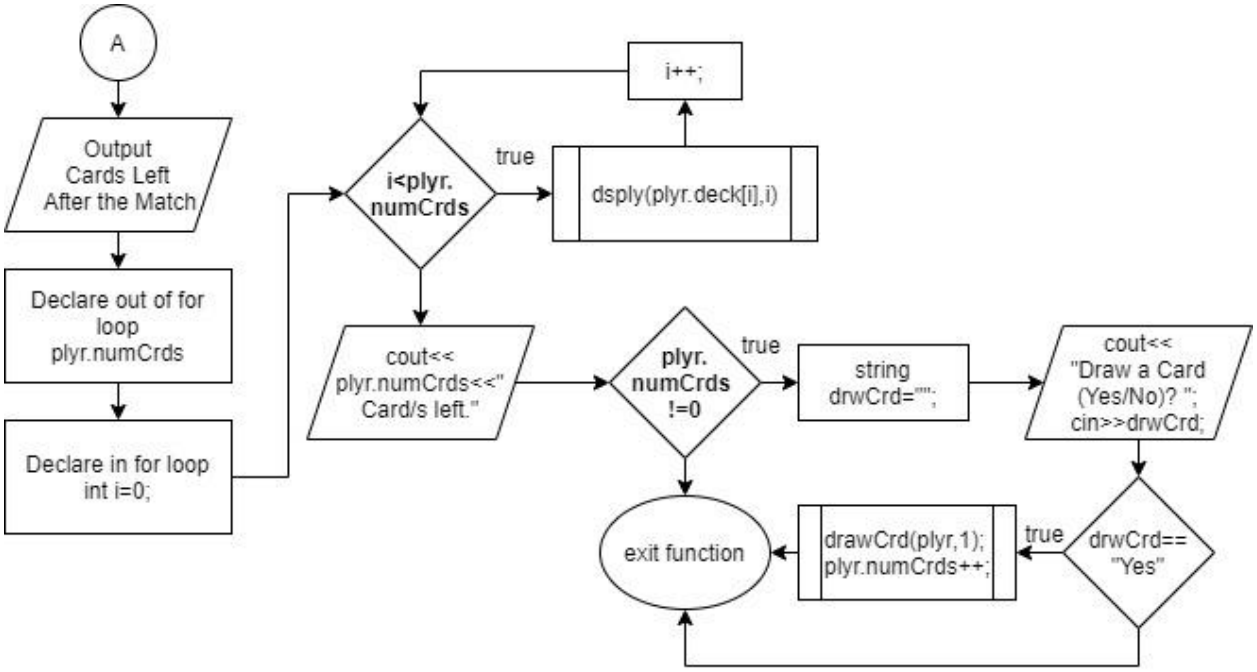
```
    //Open the bin file (UnoCards.bin) in input, output, and binary modes.
    //Using a for loop:
        //Set the cards that we need for both the Player and Opponent
        //Write the Player's cards to the txt file
        //Write the Player's cards to the bin file
    //Set the first card that needs to be matched
    //Using a do-while:
        //Go through the Player's turn.
        //If the player has 0 cards, exit the do-while
        //Go through the Opponent's turn
        //If the opponent has 0 cards, exit the do-while
    //Display the winner (The player with 0 cards left)
    //Find a random card to find in the bin file
    //Read the data on this randomly chosen card and return it
    //Display this card using our display function
    //Clean up by deleting the memory that was allocated for both players
    //Also, delete the memory that was allocated for the card read in at the end
    //Close the files we were working with
//Exit main - End of the Program
```

Project 1: Flow-Chart




void wrtBin(fstream &out,UnoCard card)
out.write(reinterpret_cast<char *>(&card.value),sizeof(unsigned char));
out.write((card.color),SIZE);
out.write((card.descr),SIZE);
exit function
UnoCard *readBin(fstream &in, int card)
UnoCard *rndm=new UnoCard;
long cursor=0L;
card !=0
true
cursor=card* (sizeof(unsigned char)+SIZE+SIZE);
in.seekg(cursor,ios::beg);
in.read(reinterpret_cast<char *>(&rndm->value),sizeof(unsigned char));
cursor+=sizeof(unsigned char);
in.seekg(cursor,ios::beg);
in.read((rndm->color),SIZE);
cursor+=SIZE;
in.seekg(cursor,ios::beg);
in.read((rndm->descr),SIZE);
return rndm
void copyCrd(UnoCard &card1,UnoCard &card2)
strcpy(card2.color,card1.color);
card2.value=card1.value;
strcpy(card2.descr,card1.descr);
strcpy(card1.color,"");
card1.value=0;
strcpy(card1.descr,"");
exit function
void drawCrd(UnoPlayer &plyr,int draw)
Declare out of for loop plyr.numCrds+draw
Declare in for loop int i=plyr.numCrds
i++;
plyr.deck[i]=setCrd()
true
i< plyr.numCrds +draw
exit function
void dsply(UnoCard card,int i)
card.value==0 ||card.value==10 ||card.value==11
true
cout<<"Card "<<i+1<<" = [" <<card.color<<" "<<setw(5) <<card.descr<<"]"<<endl;
cout<<"Card "<<i+1<<" = [" <<card.color<<" "<<setw(2) <<static_cast<int> (card.value)<<"]"<<endl;
exit function
void wrtTxt(fstream &out,UnoCard card,int i)
card.value==0 ||card.value==10 ||card.value==11
true
out<<"Card "<<i+1<<" = [" <<card.color<<" "<<setw(5) <<card.descr<<"]"<<endl;
out<<"Card "<<i+1<<" = [" <<card.color<<" "<<setw(2) <<static_cast<int> (card.value)<<"]"<<endl;
exit function


void takeTrn(UnoPlayer &plyr,UnoPlayer &oppt,UnoCard &match)
int option=0; //Card that is chosen
Display Card to Match dsply(match,0)
Declare out of for loop plyr.numCrds
Declare in for loop int i=0
i< plyr. num Crds
true
dsply(plyr.deck[i],i)
i++;
cout<< "Which card do you choose? " cin>>option
strcmp (plyr.deck[option-1].descr,"Wild Card")==0
A
true
plyr.numCrds--; char clr[SIZE];
cout<< "Choose a new color (Red/Blue/Green/Yellow <<endl<< "and type it exactly as you see it: "; cin>>clr;
strcpy(match.color,clr); copyCrd(plyr.deck[plyr.numCrds], plyr.deck[option-1]);
plyr.deck[option-1]. value==10 &&match.value==10
true
plyr.numCrds--; oppt.numCrds++;
drawCrd(oppt,1); copyCrd (plyr.deck[option-1],match); copyCrd(plyr.deck[plyr.numCrds], plyr.deck[option-1]);
A
plyr.deck[option-1] .value==10 &&strcmp (plyr.deck[option-1] color,match.color)==0
true
plyr.numCrds--; oppt.numCrds++;
drawCrd(oppt,1); copyCrd (plyr.deck[option-1],match); copyCrd(plyr.deck[plyr.numCrds], plyr.deck[option-1]);
A
plyr.deck[option-1] .value==11 && match.value==11
true
plyr.numCrds--; oppt.numCrds+=2;
drawCrd(oppt,2); copyCrd (plyr.deck[option-1],match); copyCrd(plyr.deck[plyr.numCrds], plyr.deck[option-1]);
A
plyr.deck[option-1] .value==11 &&strcmp (plyr.deck[option-1] color,match.color)==0
true
plyr.numCrds--; oppt.numCrds+=2;
drawCrd(oppt,2); copyCrd (plyr.deck[option-1],match); copyCrd(plyr.deck[plyr.numCrds], plyr.deck[option-1]);
A
strcmp (plyr.deck[option-1] .color,match.color)==0 ||plyr.deck[option-1] value==match.value
Output Can't Match The Two.
true
plyr.numCrds--;
copyCrd(plyr.deck[option-1],match); copyCrd(plyr.deck[plyr.numCrds], plyr.deck[option-1]);
A


A
Output
Cards Left
After the Match
Declare out of for
loop
plyr.numCrds
Declare in for loop
int i=0;
i<plyr.
numCrds
true
dsply(plyr.deck[i],i)
i++;
cout<<
plyr.numCrds<<"
Card/s left."
plyr.
numCrds
!=0
true
string
drwCrd="";
cout<<
"Draw a Card
(Yes/No)? ";
cin>>drwCrd;
drwCrd==
"Yes"
true
drawCrd(plyr,1);
plyr.numCrds++;
exit function

Sample Screen Shots:

This is an image of the UnoCard Structure that is being utilized throughout the program. A single card is represented as a single structure variable with a color, a value, and a description.

```
const int SIZE=10;
#ifndef UNOCARD_H
#define UNOCARD_H

struct UnoCard{
    //Card Description
    char descr[SIZE];
    //Card Color
    char color[SIZE];
    //Card's Face Value
    unsigned char value;
};

#endif /* UNOCARD_H */
```

```
#include "UnoCard.h"
#ifndef UNOPLAYER_H
#define UNOPLAYER_H

struct UnoPlayer{
    //The # of cards a player has.
    int numCrds;
    //An entire deck of cards
    UnoCard *deck;
};

#endif /* UNOPLAYER_H */
```

To the right is an image of the UnoPlayer structure. Notice that the UnoCard structure is nested within the UnoPlayer structure. Moreover, it is a pointer that holds an array of UnoCards. In the program, the UnoPlayer will store a player's deck of cards and this structure will be passed to a function in order to proceed in a turn-based fashion.

```
//Function Prototypes
UnoCard setCrd();
void dsply(UnoCard,int);              //Display a single structure variable
void copyCrd(UnoCard &,UnoCard &);    //Copy the contents of a structure variable
void drawCrd(UnoPlayer &,int);        //Draw 1 or 2 random cards and store them
void takeTrn(UnoPlayer &,UnoPlayer &,UnoCard &);
void wrtBin(fstream &,UnoCard);       //Write Binary Data (Deck of Cards)
UnoCard *readBin(fstream &,int);      //Read in the Card of choice form Bin file
void wrtTxt(fstream &,UnoCard,int);   //Write the Card to a Text file
```

Above is an image of the function prototypes. The first five are focused on gameplay while the other three are focused on reading and writing to files. Two of the latter prototypes are used in reading and writing binary data to a binary file. The last prototype was an add-in that wasn't necessary, but simply served as practice for writing data to a text file via a function.

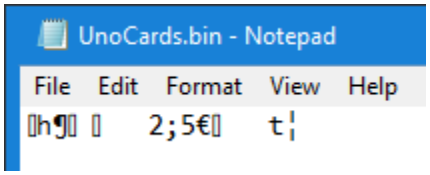
```
//For looping as long as we need to.
bool cont;
//Begin Matching Cards and Taking Turns
do{
    cont=true;
    //Player 1 Begins the Round.
    cout<<"Player's Turn"<<endl;
    takeTrn(plyr,oppt,match);
    //If Player has no cards, he wins and the game stops.
    if(plyr.numCrds==0) cont=false;
    //If Player still has cards, this will execute.
    if(cont==true){
        //Opponent's Turn Follows After.
        cout<<"Opponent's Turn"<<endl;
        takeTrn(oppt,plyr,match);
        if(oppt.numCrds==0) cont=false;
    }
    //Keep Looping Until Either Finishes Their Cards
}while(cont);
```

The image above is the code that carries out the Uno game. Since I was able to package several concepts and ideas into the function above, I was able to create a concise program in main for the game's execution. Furthermore, the code isn't too complicated, rather it is meant to be comprehensible and easy to follow along with.

```
//Deallocate the memory we set.
delete [] plyr.deck;
delete [] oppt.deck;
delete chsnCrd;
//Close the files
out.close();
bin.close();
//Exit the Program - Cleanup
return 0;
```

```
//Player's Deck of Cards
plyr.deck=new UnoCard[plyr.numCrds*3];
//Opponent's Deck of Cards
oppt.deck=new UnoCard[oppt.numCrds*3];
```

The image above is the clean up process that takes place at the end of my program. I delete whatever memory was dynamically allocated and I close all files I worked with. The image below is an example of memory being dynamically allocated using the new keyword. Near the end of the program, I also allocate memory for an UnoCard which must be deleted. In not doing so, one risks a resulting memory leak.



The image on the left is an example of how binary data is represented in binary files. This is just one card that has been written to a binary file.

This is a demonstration of a player with seven cards starting their turn. They have to match a "Yellow 3" in which they choose card 4 or a "Yellow 4". They have successfully matched the card as noted and the opponent's turn will follow after. To the right, we have an image of a quick game where players had only 1 card. At the very end, data is read from a binary file telling us what the only card we had was. Not only is the winner announced, a random card will be chosen from the very beginning to display. In this case, we only had one card to begin with.

```
Card to Match: Card 1 = [Yellow  3]
Available Cards to Choose From...
Card 1 = [None Wild Card]
Card 2 = [Red  5]
Card 3 = [Blue  4]
Card 4 = [Yellow  4]
Card 5 = [Red Draw 1]
Card 6 = [None Wild Card]
Card 7 = [Red  4]
Which card do you choose? 4

Successfully Matched.
Cards Left After the Match
Card 1 = [None Wild Card]
Card 2 = [Red  5]
Card 3 = [Blue  4]
Card 4 = [Red  4]
Card 5 = [Red Draw 1]
Card 6 = [None Wild Card]
6 Card/s left.
```

```
Eleventh Program of My Uno Project 1.

How many cards should each player start off with? 1
Player's Turn
Card to Match: Card 1 = [Red Draw 1]
Available Cards to Choose From...
Card 1 = [None Wild Card]
Which card do you choose? 1

A Wild Card was Used.
Choose a new color (Red/Blue/Green/Yellow)
and type it exactly as you see it: Red
Cards Left After the Match
0 Card/s left.


Player Wins!
Random Card to Find From When We Began: 1
Card 1 = [None Wild Card]
```

```
Card to Match: Card 1 = [Blue  4]
Available Cards to Choose From...
Card 1 = [Yellow  4]
Card 2 = [Blue Draw 1]
Card 3 = [Yellow  8]
Card 4 = [Red  3]
Card 5 = [Red  8]
Card 6 = [Red  8]
Card 7 = [Green Draw 1]
Which card do you choose? 2

Colors Matched and One Card was Drawn.
Cards Left After the Match
Card 1 = [Yellow  4]
Card 2 = [Green Draw 1]
Card 3 = [Yellow  8]
Card 4 = [Red  3]
Card 5 = [Red  8]
Card 6 = [Red  8]
6 Card/s left.
```

In this image, the player uses a Draw 1 card which forces the opponent to draw a random card. This card will be added to their deck of cards on their turn. A message is also displayed letting the player know that they used this card to force their opponent to draw a card. A Draw 2 would work the same way as the image on the left. The opponent would be forced to draw two random cards. This is achieved by a function that uses a for loop to draw as many cards as specified.

Project 1 Check-Off Sheet:

Chapter	Section	Concept	Points for Inclusion	Location in Code	Comments
9		Pointers/Memory Allocation			
	1	Memory Addresses			
	2	Pointer Variables	5	Line 106	A Pointer to UnoCard
	3	Arrays/Pointers	5	Lines 63/65	Pointers to UnoCard Arrays
	4	Pointer Arithmetic			
	5	Pointer Initialization			
	6	Comparing			
	7	Function Parameters	5	Line 68/75	Passed In Version 9
	8	Memory Allocation	5	Lines 63/65	To Allocate Arrays
	9	Return Parameters	5	Line 106	Return UnoCard Pointer
	10	Smart Pointers			
10		Char Arrays and Strings			

	1	Testing			
	2	Case Conversion			
	3	C-Strings	10	Lines 40/14/16	Line 14/16 (UnoCard.h)
	4	Library Functions			
	5	Conversion			
	6	Your own functions			
	7	Strings	10	Lines 38/289	Line 289 (In a function)
11		Structured Data			
	1	Abstract Data Types			
	2	Data			
	3	Access			
	4	Initialize			
	5	Arrays	5	Lines 63/65	UnoCard Arrays
	6	Nested	5	Lines 58/60	UnoPlayer Structures
	7	Function Arguments	5	Lines 88/95	UnoPlayers Passed In
	8	Function Return	5	Lines 74/77	Return UnoCard
	9	Pointers	5	Line 106	UnoCard Pointer
	10	Unions ****			
	11	Enumeration	5	Lines 38/46/57	Found In Version 9

12		Binary Files			
	1	File Operations			
	2	Formatting	2	Line 150-165	Display Function
	3	Function Parameters	2	Line 76	Writing to Bin File
	4	Error Testing			
	5	Member Functions	2	Line 297-318	Seekg, Read, Write Functions
	6	Multiple Files	2	Lines 38/40	Txt File + Bin File
	7	Binary Files	5	Line 40	Bin File Name
	8	Records with Structures	5	Lines 63/65/74/77	Records of Structured Data (Deck of Cards)
	9	Random Access Files	5	Lines 306-316	Jumping Cursor When Needed
	10	Input/Output Simultaneous	2	Lines 76/106	Writing + Reading Bin Data
		Total	100		